Activité pratique : Membres, héritage et redéfinition.

Exercice 1 :

1. Créez une classe appelée **Compte** avec les attributs suivants :
   - **numero** (numéro de compte, une chaîne)
   - **solde** (solde, un double)
   - **nbCompte** (nombre de comptes crées, entier, statique)

2. Ajoutez un constructeur sans paramètres à la classe **Compte** qui initialise **numero** à une chaîne vide et solde à 0. Puis incrémente **nbComptes** ;

3. Créez des getters et des setters pour les attributs **numero** et **solde**.

4. Ajoutez une méthode **deposer** qui prend un montant en paramètre et ajoute ce montant au solde du compte.

5. Ajoutez une méthode **retirer** qui prend un montant en paramètre et déduit ce montant du solde du compte, à condition que le solde soit suffisant. Si le solde n'est pas suffisant, affichez un message d'erreur.

6. Ajoutez une méthode **afficherCompteInfo** qui affiche le numéro de compte et le solde actuel.

7. Ajoutez une méthode statique **afficherNbComptes** qui affiche le nombre de comptes crées.

8. Créez un programme principal (main) pour tester la classe **Compte**. Créez un compte bancaire, effectuez des dépôts, des retraits, affichez les informations du compte à différentes étapes, et assurez-vous que les opérations se déroulent correctement.

Exercice 2 :

Vous devez implémenter une classe **MachineProduction** qui gère la production d'objets et suit un compteur de production **global** et **unique** partagé entre toutes les machines.

- **Contraintes :**

1. Chaque machine possède un **identifiant unique**.
2. Un **compteur statique** garde le nombre total d'objets produits par toutes les machines.
3. Chaque machine peut produire un certain nombre d'objets, et ce nombre s'ajoute au **compteur global**.
4. Un **plafond de production** existe et limite la production globale.
5. Une **méthode statique** permet d'accéder au compteur global.

- **Spécifications de la classe MachineProduction**

1. **Attributs d'instance :**
   - **id** : Identifiant unique de la machine.
   - **nombreProduits** : Nombre d'objets produits par cette machine.
2. **Attributs statiques :**
   - **prochainId** : Un entier statique pour attribuer des identifiants uniques aux machines.
   - **totalProduits** : Un entier statique représentant le nombre total d'objets produits.
   - **plafondProduction** : Un plafond maximum pour la production.
3. **Méthodes :**
   - **MachineProduction()**: Constructeur qui assigne un ID unique à chaque machine.
   - **static void setPlafondProduction(int plafond):** Définit le plafond de production.
   - **boolean produire(int quantite):** Permet à une machine de produire une certaine quantité, mais **uniquement si le plafond n'est pas atteint**.
   - **static int getTotalProduits():** Retourne le nombre total d'objets produits par toutes les machines.

- **Exemple d'Utilisation :**

```
public class TestMachine {
    public static void main(String[] args) {
        MachineProduction.setPlafondProduction(100);

        MachineProduction m1 = new MachineProduction();
        MachineProduction m2 = new MachineProduction();

        m1.produire(30);
        m2.produire(50);
        m1.produire(40); // Dépassera le plafond

        System.out.println("Total produits : " +
MachineProduction.getTotalProduits());
    }
}
```

Exercice 3 :

On souhaite créer une application JAVA pour la gestion des livres et des adhèrents d'une bibliothèque.

1. Créez une classe Personne avec les attributs privés : nom, prenom, email, tel, et age. Ajoutez le constructeur avec paramètres pour initialiser les différents attributs et la méthode toString() qui retourne une chaîne de caractères contenant les valeurs des attributs.

2. Créez une deuxième classe Adherent qui hérite de la classe Personne et qui contient l'attribut numAdherent et redéfinit la méthode toString().

3. Créez une troisième classe Auteur qui hérite de la classe Personne, qui contient l'attribut numAuteur et redéfinit la méthode toString().

4. Créez la classe Livre qui contient un attribut ISBN (entier), un titre (chaîne) et un auteur. Ajoutez également la méthode toString() qui retourne le ISBN, le titre et les informations de l'auteur.

5. Créez une application qui contient une méthode main() pour tester les différentes classes, dans laquelle :

— déclarez et intentiez un adhèrent ;

— déclarez et instanciez un livre qui est écrit par un auteur ;

— affichez les informations de l'adhèrent et du livre.

Exercice 4 :

Créez une classe de base appelée **Vehicule** avec les attributs et méthodes suivants :

- Attributs :
  - **nom** : Une chaîne de caractères représentant le nom du véhicule.
  - **prix** : Un double représentant le prix du véhicule.
- Méthodes :
  - **emettreSon():** Une méthode qui affiche un son générique du véhicule (par exemple, "Le véhicule émet un son inconnu.").
  - **afficherInformations():** Une méthode qui affiche des informations de base sur le véhicule, y compris son nom et son prix.

Créez des sous-classes pour différents types de véhicules, par exemple :

- **Voiture** : une classe qui hérite de **Vehicule** et a les attributs supplémentaires **modele**, **annee**, et redéfinit la méthode **emettreSon()** pour afficher "La voiture vrombit.".
- **Moto** : une classe qui hérite de **Vehicule** et a les attributs supplémentaires **marque**, **puissance**, et redéfinit la méthode **emettreSon()** pour afficher "La moto rugit.".
- **Avion** : une classe qui hérite de **Vehicule** et a les attributs supplémentaires **compagnie**, **vitesseMax**, et redéfinit la méthode **emettreSon()** pour afficher "L'avion fait un bruit de moteur puissant.".

Créez un programme principal (une classe) pour tester vos classes en créant des objets de différents types de véhicules, en définissant les attributs spécifiques à chaque véhicule, y compris le prix, et en appelant la méthode **emettreSon()** pour faire émettre le son spécifique de chaque véhicule. Enfin, affichez toutes les informations, y compris les attributs supplémentaires et le prix, avec la méthode **afficherInformations().**

Exercice 5 : Classes abstraites

On souhaite créer une application en java qui permet de gérer les salaires des ingénieurs et des managers d'une entreprise de développement informatique.

1. Créez la classe abstraite **Employe** avec les attributs **nom**, **prenom**, **email**, **telephone**, et **salaire**. Ajoutez les constructeurs avec et son paramètres, puis la méthode abstraite **calculerSalire**() qui retourne le salaire d'un employé.
2. Créez la classe **Ingénieur** avec l'attribut spécialité. Redéfinissez la méthode **calculerSalire**() sachant qu'on prévoit une augmentation de 15% par rapport à son salaire normal.
3. Créez la classe **Manager** avec l'attribut service. Redéfinissez la méthode **calculerSalire**() sachant qu'on prévoit une augmentation de 20% par rapport à son salaire normal.
4. Créez une application qui contient une méthode main() pour tester les différentes classes, dans laquelle :
   — déclarez et intentiez un ingénieur ;
   — déclarez et intentiez un manager ;
   — affichez les informations de l'ingénieur et du manager (nom, prénom, salaire, service, et spécialité)

Exercice 6 : Classe abstraites

On souhaite créer une hiérarchie de classes pour représenter différentes figures géométriques en utilisant des classes abstraites en Java. Vous devrez définir une classe abstraite **Figure** qui servira de base pour les différentes figures, telles que les cercles, les rectangles et les triangles. Chaque sous-classe de Figure devra implémenter les méthodes **calculerAire()** et **calculerPerimetre()** pour calculer l'aire et le périmètre de la figure.

1. Créez une classe abstraite **Figure** avec les caractéristiques suivantes :
   - Un attribut **protégé** pour le **nom** de la figure.
   - Une méthode abstraite **calculerAire**() pour calculer l'aire de la figure.
   - Une méthode abstraite **calculerPerimetre**() pour calculer le périmètre de la figure.

- Une méthode **afficherDetails**() pour afficher le nom de la figure, son aire et son périmètre.

Créez des sous-classes pour différentes figures géométriques (par exemple, Cercle, Rectangle, Triangle) qui étendent la classe Figure. Pour chaque sous-classe :

2. Classe **Cercle** :
   - Attributs :
     - **rayon** (double) : Le rayon du cercle.
   - Constructeur :
     - **Cercle**(String nom, double rayon) : Initialise un cercle avec un nom et un rayon.
   - Méthodes :
     - **calculerAire**() : Calcule l'aire du cercle (π * rayon^2).
     - **calculerPerimetre**() : Calcule le périmètre du cercle (2 * π * rayon).
3. Classe **Rectangle** :
   - Attributs :
     - **longueur** (double) : La longueur du rectangle.
     - **largeur** (double) : La largeur du rectangle.
   - Constructeur :
     - **Rectangle**(String nom, double longueur, double largeur) : Initialise un rectangle avec un nom, une longueur et une largeur.
   - Méthodes :
     - **calculerAire**() : Calcule l'aire du rectangle (longueur * largeur).
     - **calculerPerimetre**() : Calcule le périmètre du rectangle (2 * (longueur + largeur)).

Dans une classe principale, créez des objets de différentes figures géométriques, calculez leur aire et leur périmètre en utilisant les méthodes des sous-classes, et affichez les détails de chaque figure à l'aide de la méthode **afficherDetails**.

Exercice 7 : Les interfaces

1. Interface **Empruntable** :
   - Créez une interface appelée **Empruntable** qui définit deux méthodes : **emprunter**() et **retourner**().
   - Cette interface représente le comportement commun à tous les objets qui peuvent être empruntés et retournés.
2. Classes Concrètes **Livre** et **DVD** :
   - Implémentez l'interface **Empruntable** avec deux classes concrètes : **Livre** et **DVD**.
   - Chaque classe doit avoir des attributs spécifiques, tels que le **titre**, **l'auteur** pour le **livre**, et le **titre**, le **réalisateur** pour le **DVD**.
   - Créer un attribut booléen pour suivre l'état d'emprunt de chaque objet.

3. Classe Utilisateur :
   - Créez une classe Utilisateur avec un attribut pour le nom de l'utilisateur.
   - Ajoutez une méthode **emprunterObjet(Empruntable objet)** qui prend en paramètre un objet empruntable et appelle la méthode **emprunter()** sur cet objet.
4. Classe **GestionBibliothequeApp** :
   - Dans une classe principale appelée **GestionBibliotheque**, créez quelques instances de Livre, DVD et Utilisateur.
   - Appelez la méthode **emprunterObjet()** sur l'utilisateur en passant différentes combinaisons d'objets empruntables (**livres** et **DVD**).
   - Appelez également la méthode **retourner()** sur certains objets empruntés.

<u>Exercice 8</u> : Le polymorphisme

On souhaite développer une application pour gérer les paiement, pour cela, vous Créez les classes suivantes :

1. Classe abstraite de **Paiement** :

- Créez une classe abstraite de base appelée **Paiement** avec une méthode abstraite **effectuerPaiement(double montant).**
- La classe Paiement peut avoir des attributs communs à tous les moyens de paiement, par exemple, le **montant**, le **numéro de transaction**, etc.

2. Classes Dérivées **CarteCredit** et **PayPal** :
   - Héritez de la classe **Paiement** pour créer deux classes dérivées : **CarteCredit** et **PayPal**.
   - Chaque classe dérivée peut ajouter des attributs spécifiques, tels que le **numéro de carte** pour **CarteCredit** ou l'adresse **email** pour **PayPal**.
   - Implémentez la méthode **effectuerPaiement(double montant)** dans chaque classe dérivée pour simuler le processus de paiement en affichant un message approprié.
3. Classe **Commande** :
   - Créez une classe **Commande** qui représente une transaction d'achat.
   - Ajoutez un attribut pour le montant de la commande et un attribut pour le moyen de paiement (**CarteCredit** ou **PayPal**), de type **Paiement**.
   - Implémentez une méthode, par exemple **processPayment(),** qui utilise le polymorphisme pour effectuer le **paiement**, indépendamment du type de moyen de paiement.
4. Classe **GestionPaiementApp** :
   - Dans une classe principale appelée **GestionPaiementApp**, créez des instances de Commande avec différents moyens de paiement (**CarteCredit** et **PayPal**).
   - Appelez la méthode **processPayment()** pour chaque commande et observez comment le polymorphisme permet d'utiliser la même méthode pour différents types d'objets.